Exercise 7 solutions

June 29, 2023

Exercise 1: Theoretical Considerations

The EKF is an implementation of the Bayes Filter.

(a) *The Bayes filter processes three probability density functions, i. e., $p(x_t \mid u_t, x_{t-1})$, $p(z_t \mid x_t)$, and bel(x_t). State the normal distributions of the EKF which correspond to these probabilities.*

The assumption of the Kalman and Extended Kalman Filter is that the three stated probability density functions are normal, i.e., they are of the form:

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \det \boldsymbol{\Sigma}}} \exp\left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

Where n is the size of the random vector $\mathbf{x}$, $\boldsymbol{\mu}$ is the mean of $\mathbf{x}$, and $\boldsymbol{\Sigma}$ is its covariance matrix (an $n \times n$ symmetric positive definite matrix).

For the EKF we thus have that:

- $p(x_t \mid u_t, x_{t-1}) = g(u_t, x_{t-1}) + \epsilon_t$ is the distribution that predicts the next state from the previous one and from the control vector. If $\epsilon_t = \mathcal{N}(\epsilon_t; 0, Q_t)$ is zero-mean normally distributed, the distribution becomes $p(x_t \mid u_t, x_{t-1}) = \mathcal{N}(x_t; g(u_t, x_{t-1}), Q_t)$.
- $p(z_t \mid x_t) = h(x_t) + \delta_t$ is the distribution that predicts the measurement from the current state. If $\delta_t = \mathcal{N}(\epsilon_t; 0, R_t)$ is zero-mean normally distributed, it becomes $p(z_t \mid x_t) = \mathcal{N}(z_t; h(x_t), R_t)$.
- bel$(x_t) = \mathcal{N}(x_t; \mu_t, \Sigma_t)$ is the current estimate of the distribution of the state. Mean and covariance are given by the Kalman filter correction formulas.

Take heed of the fact that, formally speaking, $p(x_t \mid u_t, x_{t-1})$ and $p(z_t \mid x_t)$ are Gaussian in x_t only when $g(u_t, \mu_{t-1})$ and $h(\mu_t)$ are linear in x_t (technically, affine, as they may include a shift factor). In reality, this is often not the case, and as a consequence the EKF is only able to do approximate inference.

(b) *Explain in a few sentences all of the components of the EKF, i. e., μ_t, Σ_t, g, G_t, h, H_t, Q_t, R_t, K_t and why they are needed. What are the differences and similarities between the KF and the EKF?*

The Extended Kalman filter depends on the definition of the following components:

$g(u, x)$: This function estimates the current state x_t from the control and the previous state variables. In mobile robotics applications this function often defines the motion model of the system.

G_t : The Jacobian matrix of g evaluated at the previous estimate of the state, namely:

$$G_t = \left.\frac{\partial g}{\partial x}\right|_{x=\mu_{t-1}}$$

This matrix corresponds to A_t in a standard Kalman filter and accounts for the linearization approximation of the EKF.

Q_t : This is the covariance matrix (assumed to be known) of the zero-mean Gaussian error that corrupts the prediction of the state, i.e., the present state variables are given by the prediction of the function g plus some noise. This is necessary as in reality every prediction model is subject to some additional noise (e.g., imperfections in the terrain, wheel slippage, etc.).

$h(x)$: This function estimates the measurement z_t from the current state variables. In mobile robotics applications this function defines the measurement model of the system, such as range and bearing of a landmark.

H_t : The Jacobian matrix of h evaluated at the current estimate of the state from prediction only, namely:

$$H_t = \left.\frac{\partial h}{\partial x}\right|_{x=\bar{\mu}_t}$$

This matrix corresponds to C_t in a standard Kalman filter and accounts for the linearization approximation of the EKF.

R_t : This is the covariance matrix (assumed to be known) of the zero-mean Gaussian error that corrupts the measurements, i.e., the measurement is given by the function h plus some noise. This is necessary as any measurement is subject to some noise (e.g., accuracy of the sensor, systematic errors, etc.).

μ_t : This is the estimate for the mean of the distribution of the state at time t computed by the EKF. If g and h are linear and all of the additional noises are normally distributed it has been proven that the (Extended) Kalman filter will provide the optimal estimate of the mean in least squares terms.

Σ_t : This is the estimate for the covariance matrix of the distribution of the state at time t computed by the EKF. As for the mean, this is optimal under linearity assumptions.

The main difference between the Extended Kalman filter and the Kalman filter is that the latter deals only with linear models, and in doing so it provides an optimal estimate. The EKF on the other hand, is able to deal with nonlinear models by linearizing, at the cost of not being optimal. In terms of formulas, the difference is the following:

		Kalman filter	**Extended Kalman filter**
Prediction	1.	$\bar{\mu}_t = A_t\,\mu_{t-1} + B_t\,u_t$	$\bar{\mu}_t = g(u_t, \mu_{t-1})$
	2.	$\bar{\Sigma}_t = A_t\,\Sigma_{t-1} A_t^T + Q_t$	$\bar{\Sigma}_t = G_t\,\Sigma_{t-1} G_t^T + Q_t$
	3.	$K_t = \bar{\Sigma}_t\,C_t^T\left(C_t\,\bar{\Sigma}_t C_t^T + R_t\right)^{-1}$	$K_t = \bar{\Sigma}_t\,H_t^T\left(H_t\,\bar{\Sigma}_t H_t^T + R_t\right)^{-1}$
Correction	4.	$\mu_t = \bar{\mu}_t + K_t\,(z_t - C_t\,\bar{\mu}_t)$	$\mu_t = \bar{\mu}_t + K_t\,(z_t - h(\bar{\mu}_t))$
	5.	$\Sigma_t = (I - K_t\,C_t)\,\bar{\Sigma}_t$	$\Sigma_t = (I - K_t\,H_t)\,\bar{\Sigma}_t$

Notice that the only real differences are in replacing 1. and 4. with their nonlinear counterparts, and in replacing A_t and C_t with the Jacobians of g and h (resp. G_t and H_t).

Exercise 2: EKF Prediction Step

We assume a differential drive robot operating on a 2-dimensional plane, i.e., its state is defined by $\langle x, y, \theta\rangle$. Its motion model is defined on slide 10 (Odometry Model) in the chapter Probabilistic Motion Models of the lecture slides.

(a) *Derive the Jacobian matrix G_t of the noise-free motion function g. Do not use Python.*

Let us refer to $s_t = \langle x_t, y_t, \theta_t\rangle$ as the state variables in order to avoid ambiguities with the x coordinate of the robot. If we denote the control vector u_t by $\langle \delta_{r_1}, \delta_t, \delta_{r_2}\rangle$, this particular motion model is defined as:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = g(s_{t-1}, u_t) = \begin{bmatrix} x_{t-1} + \delta_t \cos(\theta_{t-1} + \delta_{r_1}) \\ y_{t-1} + \delta_t \sin(\theta_{t-1} + \delta_{r_1}) \\ \theta_{t-1} + \delta_{r_1} + \delta_{r_2} \end{bmatrix}$$

The Jacobian G_t is obtained by differentiating every row by $x_{t-1}, y_{t-1}, \theta_{t-1}$ and evaluating it at the current estimate of the state $\mu_{x,t-1}, \mu_{y,t-1}, \mu_{\theta,t-1}$:

$$G_t = \begin{array}{cccc} \frac{\partial}{\partial x_{t-1}} & \frac{\partial}{\partial y_{t-1}} & \frac{\partial}{\partial \theta_{t-1}} & \\ 1 & 0 & -\delta_t \sin(\mu_{\theta,t-1} + \delta_{r_1}) & g_1 \\ 0 & 1 & \delta_t \cos(\mu_{\theta,t-1} + \delta_{r_1}) & g_2 \\ 0 & 0 & 1 & g_3 \end{array}$$

Notice that $\mu_{x,t-1}$ and $\mu_{y,t-1}$ do not appear in the Jacobian since g is linear w.r.t. x_{t-1} and y_{t-1}.

(b) *Implement the prediction step of the EKF in the function* `prediction_step` *using your Jacobian G_t. For the noise in the motion model assume*

$$Q_t = \begin{pmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.02 \end{pmatrix}.$$

```python
def prediction_step(odometry, mu, sigma):
    # Updates the belief, i.e., mu and sigma, according to the motion
    # model
    #
    # mu: 3x1 vector representing the mean (x,y,theta) of the
    #     belief distribution
    # sigma: 3x3 covariance matrix of belief distribution

    x = mu[0]
    y = mu[1]
    theta = mu[2]

    delta_rot1 = odometry['r1']
    delta_trans = odometry['t']
    delta_rot2 = odometry['r2']

    #motion noise
    Q = np.array([[0.2, 0.0, 0.0],\
                 [0.0, 0.2, 0.0],\
                 [0.0, 0.0, 0.02]])

    #noise free motion
    x_new = x + delta_trans * np.cos(theta + delta_rot1)
    y_new = y + delta_trans * np.sin(theta + delta_rot1)
    theta_new = theta + delta_rot1 + delta_rot2

    #Jakobian of g with respect to the state
    G = np.array([[1.0, 0.0, -delta_trans * np.sin(theta + delta_rot1)],\
                  [0.0, 1.0, delta_trans * np.cos(theta + delta_rot1)],\
                  [0.0, 0.0, 1.0]])

    #new mu and sigma
    mu = [x_new, y_new, theta_new]
    sigma = np.dot(np.dot(G,sigma),np.transpose(G)) + Q
```

Exercise 3: EKF Correction Step

(a) *Derive the Jacobian matrix H_t of the noise-free measurement function h of a range-only sensor. Do not use Python.*

Let $l = \langle l_x, l_y \rangle$ denote the landmark expressed w.r.t. the world reference frame, whose distance we are measuring. Then the measurement function h for a single landmark is scalar (since it returns a single range value) and is as follows:

$$z_t = h(s_t, l) = \sqrt{(x_{t-1} - l_x)^2 + (y_{t-1} - l_y)^2}$$

Where we added an l argument to h in order to make it parametric with respect to the landmark that we are measuring.

Again, we compute the Jacobian H_t by differentiating w.r.t. the state variables and evaluating it w.r.t. the current estimate of the state:

$$H_t = \begin{bmatrix} \dfrac{\bar{\mu}_{x,t} - l_x}{h(\bar{\mu}_t, l)} & \dfrac{\bar{\mu}_{y,t} - l_y}{h(\bar{\mu}_t, l)} & 0 \end{bmatrix}$$

If multiple measurements are considered together, say k measurements, then we simply stack the z_t values and the H_t Jacobian matrices as follows:

$$\tilde{z}_t = \tilde{h}(s_t) = \begin{bmatrix} h(s_t, l^{(1)}) \\ h(s_t, l^{(2)}) \\ \vdots \\ h(s_t, l^{(k)}) \end{bmatrix}$$

$$\tilde{H}_t = \begin{bmatrix} \dfrac{\bar{\mu}_{x,t} - l_x^{(1)}}{h(\bar{\mu}_t, l^{(1)})} & \dfrac{\bar{\mu}_{y,t} - l_y^{(1)}}{h(\bar{\mu}_t, l^{(1)})} & 0 \\ \dfrac{\bar{\mu}_{x,t} - l_x^{(2)}}{h(\bar{\mu}_t, l^{(2)})} & \dfrac{\bar{\mu}_{y,t} - l_y^{(2)}}{h(\bar{\mu}_t, l^{(2)})} & 0 \\ \vdots & \vdots & \vdots \\ \dfrac{\bar{\mu}_{x,t} - l_x^{(k)}}{h(\bar{\mu}_t, l^{(k)})} & \dfrac{\bar{\mu}_{y,t} - l_y^{(k)}}{h(\bar{\mu}_t, l^{(k)})} & 0 \end{bmatrix}$$

Where we denoted by $l^{(i)} = \langle l_x^{(i)}, l_y^{(i)} \rangle$, the i-th landmark we are measuring.

Note that, although h does not take at all into account the orientation of the robot, we will, nevertheless, be able to estimate it. This is because the motion model makes the position and the orientation of the robot correlated, hence if we have some information on the location of the robot we invariably also have some on the orientation of the robot, which gets considered in the computation of the Kalman gain K_t. The inability to measure directly the orientation of the robot does, however, have as a drawback that its estimate will have a large variance (very noisy).

(b) *Implement the correction step of the EKF in the function* `correction_step` *using your Jacobian H_t. For the noise in the sensor model assume that R_t is the diagonal square matrix*

$$R_t = \begin{pmatrix} 0.5 & 0 & 0 & \dots \\ 0 & 0.5 & 0 & \dots \\ 0 & 0 & 0.5 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \in \mathbb{R}^{size(z_t) \times size(z_t)}.$$

```
def correction_step(sensor_data, mu, sigma, landmarks):
    # updates the belief, i.e., mu and sigma, according to the
    # sensor model
    #
    # The employed sensor model is range-only
    #
    # mu: 3x1 vector representing the mean (x,y,theta) of the
    #     belief distribution
    # sigma: 3x3 covariance matrix of belief distribution

    x = mu[0]
    y = mu[1]
    theta = mu[2]

    #measured landmark ids and ranges
    ids = sensor_data['id']
    ranges = sensor_data['range']

    # Compute the expected range measurements for each landmark.
    # This corresponds to the function h
    H = []
    Z = []
    expected_ranges = []

    for i in range(len(ids)):

        lm_id = ids[i]
        meas_range = ranges[i]

        lx = landmarks[lm_id][0]
        ly = landmarks[lm_id][1]

        #calculate expected range measurement
        range_exp = np.sqrt( (lx - x)**2 + (ly - y)**2 )

        #compute a row of H for each measurement
        H_i = [(x - lx)/range_exp, (y - ly)/range_exp, 0]
        H.append(H_i)
        Z.append(ranges[i])
        expected_ranges.append(range_exp)

    # noise covariance for the measurements
    R = 0.5 * np.eye(len(ids))

    # Kalman gain
    K_help = np.linalg.inv(np.dot(np.dot(H,sigma),np.transpose(H)) + R)
    K = np.dot(np.dot(sigma,np.transpose(H)),K_help)
    # Kalman correction of mean and covariance
    mu = mu + np.dot(K,(np.array(Z) - np.array(expected_ranges)))
    sigma = np.dot(np.eye(len(sigma)) - np.dot(K,H),sigma)
```